Extending Interfaces

Interfaces can be extended, similar to classes, using the extends keyword.

On this slide, I show an interface, OrbitEarth, that extends the FlightEnabled interface.

This interface requires all classes to implement both the OrbitEarth, and the FlightEnabled abstract methods.

```
interface OrbitEarth extends FlightEnabled {}
```

Unlike a class, an interface can use the extends expression with multiple interfaces:

```
interface OrbitEarth extends FlightEnabled, Trackable {}
```

Implements is invalid on an interface

An interface doesn't implement another interface, so the code on this slide won't compile.

In other words, implements is an invalid clause in an interface declaration.

```
interface OrbitEarth implements FlightEnabled {}  // INVALID, implements is
                                                   // invalid clause for
                                                   // interfaces
```

Abstracted Types - Coding to an Interface

Both interfaces and abstract classes are **abstracted reference types**.

Reference types can be used in code, as variable types, method parameters, and return types, list types, and so on.

When you use an abstracted reference type, this is referred to as **coding to an interface**.

This means your code doesn't use specific types, but more generalized ones, usually an interface type.

This technique is preferred, because it allows many runtime instances of various classes, to be processed uniformly, by the same code.

It also allows for substitutions of some other class or object, that still implements the same interface, without forcing a major refactor of your code.

Using interface types as the reference type, is considered a best practice.

Coding to an Interface

Coding to an interface scales well, to support new subtypes, and it helps when refactoring code.

The downside though, is that alterations to the interface may wreak havoc, on the client code.

Imagine that you have 50 classes using your interface, and you want to add an extra abstract method, to support new functionality.

As soon as you add a new abstract method, all 50 classes won't compile.

Your code isn't backwards compatible, with this kind of change to an interface.

Interfaces haven't been easily extensible in the past.

But Java has made several changes to the Interface type over time, to try to address this last problem.